

1

**SELF-DESCRIBING FILE SYSTEM****CROSS REFERENCE TO RELATED APPLICATIONS**

This application is related to, and claims the priority of, U.S. Provisional patent application Ser. No. 60/157,777 filed Oct. 5, 1999 and entitled "Self-Describing File System" by the same inventor Stephen A. Rago.

**BACKGROUND OF THE INVENTION****1. Field of the Invention**

The invention relates to the shared access of a computer system's file system storage by disparate, possibly unrelated, applications, such as portable file system administrative tools or sharing file systems in a storage area network (SAN).

**2. Description of Related Art**

A "file system" is an abstraction a computer operating system uses to ease the management of its user's data. Data are separated into storage units called "files" based on subject matter. Related files can be grouped together (usually also by subject matter) by listing their names in the same "directory."

Applications that need to read or write files do so through a "file system driver." The driver translates an application's request into the operations needed to read or write the storage locations that contain the data. The storage medium is usually some sort of magnetic or optical disk, but need not be limited to disks. For example, a file system driver can use RAM as the backing store for temporary file storage.

Applications usually don't know how their data are stored on disk, and don't want to know, for that matter. It is much better to isolate the knowledge of the file system format in some external place (the driver, in this case) than to embed it in each application. This makes the applications smaller, easier to write, and more portable. The benefits of portability are not to be underestimated. Many different file system formats exist, and it would be next to impossible to embed knowledge about each one in an application.

In addition to portability, centralizing the control in the file system provides a convenient way to serialize access to the on-disk data structures. If each application were to attempt to manage the file system data structures on disk, they would need to agree amongst themselves so that only one application modifies the same on-disk structure at a time. The file system driver relieves applications from having to worry about this task.

Thus, applications have evolved to ignore, for the most part, how their files are stored on disk. Nonetheless, there are still some cases where applications need to understand the on-disk file system format. Obviously, the tools used to create a file system or check a file system's consistency need to understand it's format. They are implicitly tied to the file system format, but other, more generic, applications might also need to be able to interpret the file system on-disk data structures.

For example, consider a conventional backup application that relies on the file system driver to interpret the file system format. The backup application searches the file system to copy all files to some backup medium. As each file is read, the file's access time is updated. This interferes with attempts to identify files that haven't been used for long periods of time. An administrator might wish to archive the stale files and remove them from the disk, since they are

2

taking up disk space that might otherwise be available to store files that are accessed more frequently.

The backup application makes this difficult to do. Of course, the backup application could save the access time before reading a file, and then restore the access time after it has finished copying the file to the backup medium. However, what if someone other than the backup application reads the file while it is being backed up? The step of restoring the access time can wipe out the change to the access time that occurred when the file was read by someone else. This can lead to the file being archived prematurely.

A possible solution is to have the backup application read the disk device containing the file system and interpret the file system data structures. This avoids the updated access time, but makes the backup application specific to this file system format. A software vendor wants to write the backup application once and avoid customizing it for each different file system format.

Although others have created self-describing files, no one has attempted to create a self-describing file system. U.S. Pat. No. 5,640,559 describes a way to encode file data and relationships among data in a self-describing format to allow them to be transmitted between computers more efficiently. Another example is the Hierarchical Data Format (HDF) defined by the National Center for Supercomputing Applications (NCSA). See NCSA *HDF5 Reference Manual*, Release 1.2, October 1999. It is a data format specification and a set of libraries used to create self-describing data files. It is commonly used to store scientific data.

Self-describing files have been used in a wide variety of applications including encoding data for communication between computer systems (U.S. Pat. No. 5,257,369), encoding the data in a storage dump (U.S. Pat. No. 5,761,739), a self-describing database management system (U.S. Pat. No. 5,857,195), storing the state of objects in object-oriented systems (U.S. Pat. No. 5,905,987), and encoding file objects in a distributed computing environment (U.S. Pat. No. 5,768,532). Before it was acquired by Microsoft, Entropic, Inc. produced a library that encoded speech files in a self-describing way. Although their documentation referred to the "ESPS File System," theirs was a library that could create a set of files, and was not a general-purpose file system as previously described. For more information see Entropic Research Laboratory, Inc. *ESPS/waves+ with EnSig™ Application Notes*. Chapter entitled "Non-ESPS Programs and the ESPS File System." Release 5.3, 1998. <http://www.ling.ed.ac.uk/help/entropic>.

U.S. Pat. No. 5,950,203 describes a system with improved access to data stored on a peripheral device. This applies to computer systems that can access the same storage resources on a Storage Area Network (SAN). In the method disclosed, the server is the entity that determines a file's block list. In a self-describing file system, however, the clients can determine the block list themselves.

Overall, the prior art does not appear to suggest or describe a system capable of attaining the same levels of portability and efficiency as the self-describing file system described herein.

**SUMMARY OF THE INVENTION**

Briefly described, the invention comprises a disk containing a file system and one or more computer systems that can access the disk. Along with the file system, the disk contains a formal description that allows applications to understand the format of the file system.